

Midterm Exam

NAME:

March 4

If you have any questions about a problem, quietly come to the front of the classroom and ask me. To pace yourself, you should allow on more than 1 minute/point. For example, on a 10 point problem, don't spend more than 10 minutes. Good Luck.

1. (10 points) Give tight asymptotic bounds for $T(n)$ in each of the following recurrences (i.e. it's sufficient to use Θ notation so use the master method whenever you can). You can assume that $T(1) = \Theta(1)$. As part of showing your work, you are required to give the value of " ℓ " and " k ". You are welcome to do the rest in your head if you want but the more you show the more we can give partial credit if you made a mistake.

(a) $T(n) = T(9n/10) + \Theta(1)$

(b) $T(n) = 2T(n/4) + 10 \ln n + 20$

(c) $T(n) = 4T(n/3) + 15n\sqrt{n} + 10n \log_3 n$ (Note: $\log_3 4 \approx 1.26$)

(d) $T(n) = 3T(n/3) + 2^{\log_2 n}$

Take each of the four answers above and put each one in one of the blanks below so that the resulting statement is true.

_____ = O (_____) = O (_____) = O (_____)

2. (5 points) Consider the following search algorithm:

```
int search(A){
    n = A.length;
    for (int i = 0; i < n; i++){
        if (A[i] == x)
            return i;
    }
    return -1;
}
```

You are to compute the expected number of times the line “if ($A[i] == x$)” is executed for array A of 7 elements when:

array position	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	x not in A
probability x in that position	.05	.15	.1	.15	.05	.3	.1	.1

While the arithmetic should be easy to do without a calculator you are welcome to just leave your answer in a form like $5 \times .4 + 2 \times .1 + \dots$.

3. (3 pts) Consider the problem of sorting n numbers, each of which is $2 \log n$ bits long.

(a) Suppose we use radix sort, treating each bit of the number as a radix sort digit (so radix sort runs counting sort $2 \log n$ times). What is the asymptotic time complexity?

(b) (3 pts) Suppose we again use radix sort, but we treat each $2 \log n$ bit number as a single digit (so radix sort runs counting sort only once). What is the asymptotic time complexity?

(c) (9 pts) How can we use radix sort to sort the n numbers in expected $O(n)$ time?

4. Consider the following problem. You are given arrays A and B which each have n elements. The elements in the arrays are **not** necessarily in sorted order, and each array can contain repeated elements. We define A and B to be *disjoint* if they contain no elements in common. For example, the arrays $[1, 7, 2, 5, 2]$ and $[6, 7, 8, 9, 6]$ are not disjoint because they both contain the number 7. However, the arrays $[2, 1, 7, 2, 5]$ and $[8, 8, 9, 6, 9]$ are disjoint.

In the *array disjointness problem*, the problem is to determine whether arrays A and B are *disjoint*. If the arrays are disjoint the algorithm should return a pair (i, j) such that $A[i] = B[j]$. Otherwise, the algorithm should return `null`.

- (a) (5 points) The following is an algorithm for solving the array disjointness problem.

```
for (int i=0; i< n; i++)
    for (int j=0; j< n; j++)
        if (A[i] == B[j]) return (i,j);
return null;
```

What is the asymptotic time complexity of this algorithm? (No explanation needed.)

- (b) (10 points) The following is another algorithm for solving the array disjointness problem, where the initial call made is `isDisjoint(A,0,n-1,B,0,n-1)`.

```
isDisjoint(A,a1,a2,B,b1,b2){
    if ((a1 > a2) || (b1 > b2)) return null; //no elements in A or B
    if ((a1==a2) && (b1==b2)) //one element in A and B
        if (A[a1] == B[b1]) return (a1,b1); //if equal return the indices
        else return null; //else they are disjoint
    amid = (int) (a1+a2)/2;
    bmid = (int) (b1+b2)/2;
    ans1 = isDisjoint(A,a1,amid,B,b1,bmid); //recurse on both left halves,
    ans2 = isDisjoint(A,amid+1,a2,B,b1,bmid); //on right of A and left of B,
    ans3 = isDisjoint(A,a1,amid,B,bmid+1,b2); //on left of A and right of B,
    ans4 = isDisjoint(A,amid+1,a2,B,bmid+1,b2); //on both right halves
    if (ans1 != null) return ans1;
    else if (ans2 != null) return ans2;
    else if (ans3 != null) return ans3;
    else if (ans4 != null) return ans4;
    else return null;
}
```

Give a recurrence equation for the time complexity of `isDisjoint` when A and B both have n elements. Then use the recurrence relation to determine the asymptotic time complexity of `isDisjoint`.

(c) (10 points) Briefly but clearly describe an algorithm for solving the array disjointness problem that has worst-case $O(n \log n)$ asymptotic time complexity.

(d) (10 points) Briefly but clearly describe an algorithm that solves the array disjointness problem using a hash table that has $O(n)$ expected time complexity. Your solution should indicate what size to make the hash table. Also, for any insert (put) operation you must clearly indicate what is used as the key, and what is used as the data.

Give a very brief time complexity analysis of your algorithm to convince us that it runs in $O(n)$ expected time.

- (e) (10 pts) Using the decision tree lower bound technique to give a lower bound on the number of comparisons used in the worst-case by any comparison-based algorithm to solve the disjointness problem. You should not be concerned if the lower bound you achieve is much lower than the time complexity of your algorithm. Just give the best lower bound that you can in 10 minutes that uses the decision tree technique. Be sure to clearly explain any claim you make that is specific to this problem.

Have a great spring break! See you when you get back.

Problem	Points Possible	Points Received
1	10	
2	5	
3	15	
4a,b	15	
4c,d	20	
4e	10	
total	75	